A Weakly Overlapping Parallel Domain Decomposition Preconditioner for the Finite Element Solution of Convection-Dominated Problems in Three Dimensions

Peter K. Jimack[a*] Sarfraz A. Nadeem[a†]

[a]Computational PDEs Unit, School of Computing, University of Leeds, LS2 9JT, UK

In this paper we describe the parallel application of a novel two level additive Schwarz preconditioner to the stable finite element solution of convection-dominated problems in three dimensions. This is a generalization of earlier work, [2,6], in 2-d and 3-d respectively. An algebraic formulation of the preconditioner is presented and the key issues associated with its parallel implementation are discussed. Some computational results are also included which demonstrate empirically the optimality of the preconditioner and its potential for parallel implementation.

1. INTRODUCTION

Convection-diffusion equations play a significant role in the modeling of a wide variety of fluid flow problems. Of particular challenge to CFD practitioners is the important case where the convection term is dominant and so the resulting flow contains small regions of rapid change, such as shocks or boundary layers. This paper will build upon previous work of [1,2,6] to produce an efficient parallel domain decomposition (DD) preconditioner for the adaptive finite element (FE) solution of convection-dominated elliptic problems of the form

$$-\varepsilon \underline{\nabla}^2 u + \underline{b} \cdot \underline{\nabla} u = f \qquad \text{on } \Omega \subset \Re^3, \tag{1}$$

where $0 < \varepsilon << \|\underline{b}\|$, subject to well-posed boundary conditions. An outline of the parallel solution strategy described in [1] is as follows.

1. Obtain a finite element solution of (1) on a coarse mesh of tetrahedra and obtain corresponding *a posteriori* error estimates on this mesh.

2. Partition Ω into p subdomains corresponding to subsets of the coarse mesh, each subset containing about the same total (approximate) error (hence some subdomains will contain many more coarse elements than others if the *a posteriori* error estimate varies significantly throughout the domain). Let processor i ($i = 1, ..., p$) have a copy of the entire coarse mesh and sequentially solve the *entire* problem using adaptive refinement only in subdomain i (and its immediate neighbourhood): the target number of elements on each processor being the same.

*Corresponding author: pkj@comp.leeds.ac.uk
†Funded by the Government of Pakistan through a Quaid-e-Azam scholarship.

3. A global fine mesh is defined to be the union of the refined subdomains (with possible minor modifications near subdomain interfaces, to ensure that it is conforming), although it is never explicitly assembled.

4. A parallel solver is now required to solve this distributed (well load-balanced) problem.

This paper will describe a solver of the form required for the final step above, although the solver may also be applied independently of this framework. The work is a generalization and extension of previous research in two dimensions, [2], and in three dimensions, [6]. In particular, for the case of interest here, where (1) is convection dominated, a stabilized FE method is required and we demonstrate that the technique introduced in [2,6] may still be applied successfully. The following section of this paper provides a brief introduction to this preconditioning technique, based upon what we call a *weakly overlapping* domain decomposition, and Section 3 presents a small number of typical computational results. The paper concludes with a brief discussion.

2. THE WEAKLY OVERLAPPING DOMAIN DECOMPOSITION PRECONDITIONER

The standard Galerkin FE discretization of (1) seeks an approximation u_h to u from a finite element space $\mathcal{S}_h$ such that

$$\varepsilon \int_\Omega \underline{\nabla} u_h \cdot \underline{\nabla} v \; d\underline{x} + \int_\Omega (\underline{b} \cdot \underline{\nabla} u_h) v \; d\underline{x} = \int_\Omega f v \; d\underline{x} \tag{2}$$

for all $v \in \mathcal{S}_h$ (disregarding boundary conditions for simplicity). Unless the mesh is sufficiently fine this is known to be unstable when $0 < \varepsilon << \|\underline{b}\|$ and so we apply a more stable FE method such as the streamline-diffusion algorithm (see, for example, [7] for details). This replaces v in (2) by $v + \alpha \underline{b} \cdot \underline{\nabla} v$ to yield the problem of finding $u_h \in \mathcal{S}_h$ such that

$$\varepsilon \int_\Omega \underline{\nabla} u_h \cdot \underline{\nabla}(v + \alpha \underline{b} \cdot \underline{\nabla} v) \; d\underline{x} + \int_\Omega (\underline{b} \cdot \underline{\nabla} u_h)(v + \alpha \underline{b} \cdot \underline{\nabla} v) \; d\underline{x} = \int_\Omega f (v + \alpha \underline{b} \cdot \underline{\nabla} v) \; d\underline{x} \tag{3}$$

for all $v \in \mathcal{S}_h$. In general α is chosen to be proportional to the mesh size h and so, as the mesh is refined, the problem (3) approaches the problem (2).

Once the usual local FE basis is defined for the space $\mathcal{S}_h$, the system (3) may be written in matrix notation as

$$A\underline{u} = \underline{b} \; . \tag{4}$$

If the domain Ω is partitioned into two subdomains (the generalization to p subdomains is considered below), using the approach described in Section 1 for example, then the system (4) may be written in block-matrix notation as

$$\begin{bmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ C_1 & C_2 & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{f}_1 \\ \underline{f}_2 \\ \underline{f}_s \end{bmatrix} \; . \tag{5}$$

Here $\underline{u}_i$ is the vector of unknown finite element solution values at the nodes strictly inside subdomain i ($i = 1, 2$) and $\underline{u}_s$ is the vector of unknown values at the nodes on the interface between subdomains. The blocks A_i, B_i, C_i and $\underline{f}_i$ represent the components of the FE system that may be assembled (and stored) independently on processor i ($i = 1, 2$). Furthermore, we may express

$$A_s = A_{s(1)} + A_{s(2)} \quad \text{and} \quad \underline{f}_s = \underline{f}_{s(1)} + \underline{f}_{s(2)} \,, \tag{6}$$

where $A_{s(i)}$ and $\underline{f}_{s(i)}$ are the components of A_s and $\underline{f}_s$ respectively that may be calculated (and stored) independently on processor i.

The system (5) may be solved using an iterative technique such as preconditioned GMRES (see [10] for example). Traditional parallel DD solvers typically take one of two forms: either applying block elimination to (5) to obtain a set of equations for the interface unknowns $\underline{u}_s$ (e.g. [5]), or solving the complete system (5) in parallel (e.g. [3]). The weakly overlapping approach that we take is of the latter form. Apart from the application of the preconditioner, the main computational steps required at each GMRES iteration are a matrix-vector multiplication and a number of inner products. Using the above partition of the matrix and vectors it is straightforward to perform both of these operations in parallel with a minimal amount of interprocessor communication (see [4] or [5] by way of two examples). The remainder of this section therefore concentrates on an explanation of our novel DD preconditioner.

Our starting point is to assume that we have two meshes of the same domain which are hierarchical refinements of the same coarse mesh. Mesh 1 has been refined heavily in subdomain 1 and in its immediate neighbourhood (any element which touches the boundary of a subdomain is defined to be in that subdomain's immediate neighbourhood), whilst mesh 2 has been refined heavily in subdomain 2 and its immediate neighbourhood. Hence, the overlap between the refined regions on each processor is restricted to a single layer at each level of the mesh hierarchy. Figure 1 shows an example coarse mesh, part of the final mesh and the corresponding meshes on processors 1 and 2 in the case where the final mesh is a uniform refinement (to 2 levels) of the initial mesh of 768 tetrahedral elements. Throughout this paper we refine a tetrahedron by bisecting each edge and producing 8 children. Special, temporary, transition elements are also used to avoid "hanging nodes" when neighbouring tetrahedra are at different levels of refinement. See [11] for full details of this procedure.

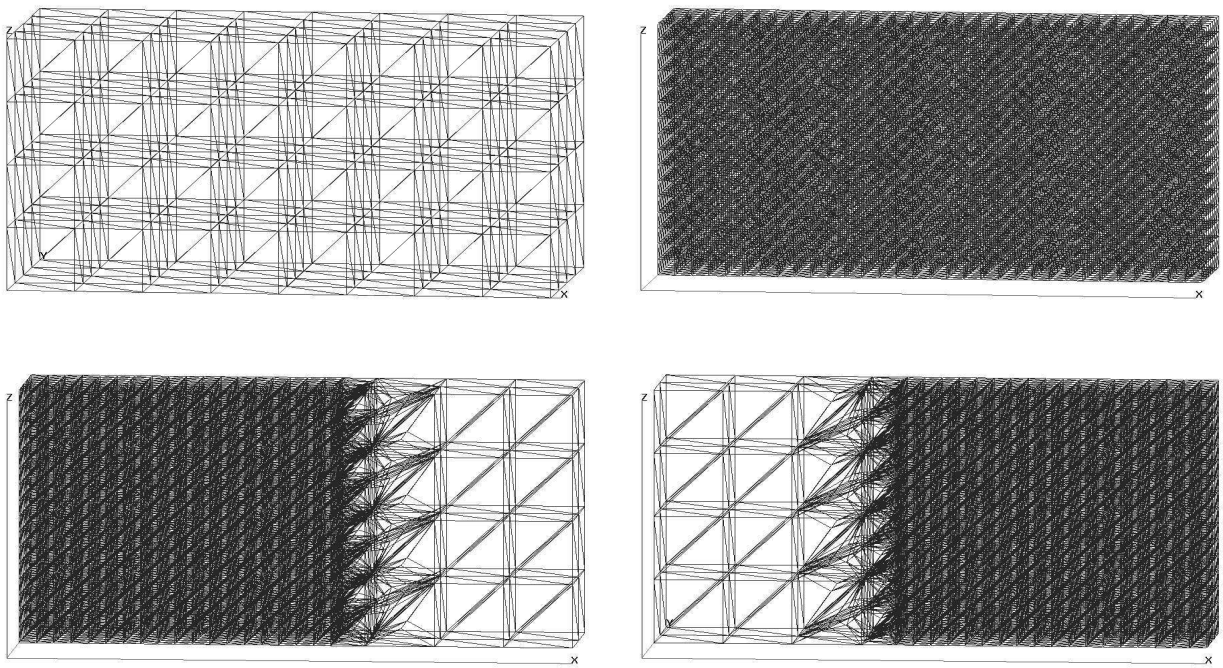
The DD preconditioner, P say, that we use with GMRES when solving (5) may be described in terms of the computation of the action of $\underline{z} = P^{-1}\underline{p}$. On processor 1 solve the system

$$\begin{bmatrix} A_1 & 0 & B_1 \\ 0 & \tilde{A}_2 & \tilde{B}_2 \\ C_1 & \tilde{C}_2 & A_s \end{bmatrix} \begin{bmatrix} \underline{z}_{1,1} \\ \underline{z}_{2,1} \\ \underline{z}_{s,1} \end{bmatrix} = \begin{bmatrix} \underline{p}_1 \\ M_2\underline{p}_2 \\ \underline{p}_s \end{bmatrix} \,, \tag{7}$$

and on processor 2 solve the system

$$\begin{bmatrix} \tilde{A}_1 & 0 & \tilde{B}_1 \\ 0 & A_2 & B_2 \\ \tilde{C}_1 & C_2 & A_s \end{bmatrix} \begin{bmatrix} \underline{z}_{1,2} \\ \underline{z}_{2,2} \\ \underline{z}_{s,2} \end{bmatrix} = \begin{bmatrix} M_1\underline{p}_1 \\ \underline{p}_2 \\ \underline{p}_s \end{bmatrix} \,, \tag{8}$$

Figure 1. An initial mesh of 768 tetrahedral elements (top left) refined uniformly into 49152 elements (top right) and the corresponding meshes on processor 1 (bottom left) and processor 2 (bottom right).



then set

$$\begin{bmatrix} \underline{z}_1 \\ \underline{z}_2 \\ \underline{z}_s \end{bmatrix} = \begin{bmatrix} \underline{z}_{1,1} \\ \underline{z}_{2,2} \\ \frac{1}{2}(\underline{z}_{s,1} + \underline{z}_{s,2}) \end{bmatrix} . \tag{9}$$

In the above notation, the blocks $\tilde{A}_2$, $\tilde{B}_2$ and $\tilde{C}_2$ (resp. $\tilde{A}_1$, $\tilde{B}_1$ and $\tilde{C}_1$) are the assembled components of the stiffness matrix for the part of the mesh on processor 1 (resp. 2) that covers subdomain 2 (resp. 1). These may be computed and stored without communication. Moreover, because of the single layer of overlap in the refined regions of the meshes, A_s may be computed and stored on each processor without communication. Finally, the rectangular matrix M_1 (resp. M_2) represents the restriction operator from the fine mesh covering subdomain 1 (resp. 2) on processor 1 (resp. 2) to the coarser mesh covering subdomain 1 (resp. 2) on processor 2 (resp. 1). This is the usual hierarchical restriction operator that is used in most multigrid algorithms (see, for example [9]).

The generalization of this idea from 2 to p subdomains is straightforward. We will assume for simplicity that there is a one-to-one mapping between subdomains and processors. Each processor, i say, produces a mesh which covers the whole domain (the coarse mesh) but is refined only in subdomain i, Ω_i say, and its immediate neighbourhood. Again, this means that the overlapping regions of refinement consist of one layer

of elements at each level of the mesh. For each processor i the global system (4) may be written as

$$\begin{bmatrix} A_i & 0 & B_i \\ 0 & \bar{A}_i & \bar{B}_i \\ C_i & \bar{C}_i & A_{i,s} \end{bmatrix} \begin{bmatrix} \underline{u}_i \\ \underline{\bar{u}}_i \\ \underline{u}_{i,s} \end{bmatrix} = \begin{bmatrix} \underline{f}_i \\ \underline{\bar{f}}_i \\ \underline{f}_{i,s} \end{bmatrix} , \tag{10}$$

where now $\underline{u}_i$ is the vector of finite element unknowns strictly inside Ω_i, $\underline{u}_{i,s}$ is the vector of unknowns on the interface of Ω_i and $\underline{\bar{u}}_i$ is the vector of unknowns (in the global fine mesh) outside of $\bar{\Omega}_i$. Similarly, the blocks A_i, B_i, C_i and $\underline{f}_i$ are all computed from the elements of the mesh inside subdomain i, etc.

The action of the preconditioner ($\underline{z} = P^{-1}\underline{p}$), in terms of the computations required on each processor i, is therefore as follows.
(i) Solve

$$\begin{bmatrix} A_i & 0 & B_i \\ 0 & \tilde{A}_i & \tilde{B}_i \\ C_i & \tilde{C}_i & A_{i,s} \end{bmatrix} \begin{bmatrix} \underline{z}_i \\ \underline{\tilde{z}}_i \\ \underline{z}_{i,s} \end{bmatrix} = \begin{bmatrix} \underline{p}_i \\ \bar{M}_i \underline{\bar{p}}_i \\ \underline{p}_{i,s} \end{bmatrix} . \tag{11}$$

(ii) Replace each entry of $\underline{z}_{i,s}$ with the average value over all corresponding entries of $\underline{z}_{j,s}$ on neighbouring processors j.
In (11) $\tilde{A}_i$, $\tilde{B}_i$ and $\tilde{C}_i$ are the components of the stiffness matrix for the mesh stored on processor i (this is not the global fine mesh but the mesh actually generated on processor i) which correspond to nodes outside of $\bar{\Omega}_i$. The rectangular matrix $\bar{M}_i$ represents the hierarchical restriction operator from the global fine mesh outside of $\bar{\Omega}_i$ to the mesh on processor i covering the region outside of $\bar{\Omega}_i$.

The main parallel implementation issue that now needs to be addressed is that of computing these hierarchical restrictions, $\bar{M}_i\underline{\bar{p}}_i$, efficiently at each iteration. Because each processor works with its own copy of the coarse mesh (which is locally refined) processor i must contribute to the restriction operation $\bar{M}_j\underline{\bar{p}}_j$ for each $j \neq i$, and processor j must contribute to the calculation of $\bar{M}_i\underline{\bar{p}}_i$ (for each $j \neq i$). To achieve this, processor i restricts its fine mesh vector $\underline{p}_i$ (covering Ω_i) to the part of the mesh on processor j which covers Ω_i (received initially from j in a setup phase) and sends this restriction to processor j (for each $j \neq i$). Processor i then receives from each other processor j the restriction of the fine mesh vector $\underline{p}_j$ (covering Ω_j on processor j) to the part of the mesh on processor i which covers Ω_j. These received vectors are then combined to form $\bar{M}_i\underline{\bar{p}}_i$ before (11) is solved. The averaging of the $\underline{z}_{i,s}$ in step (ii) above requires only local neighbour-to-neighbour communication.

3. COMPUTATIONAL RESULTS

All of the results presented in this section were computed with an ANSI C implementation of the above algorithm using the MPI communication library, [8], on a shared memory SG Origin2000 computer. The NUMA (non-uniform memory access) architecture of this machine means that timings for a given calculation may vary significantly between runs (depending on how the memory is allocated), hence all timings quoted represent the best time that was achieved over numerous repetitions of the same computation.

Table 1
The performance of the proposed DD algorithm using the stabilized FE discretization of the convection-diffusion test problem for two choices of ε: figures quoted represent the number of iterations required to reduce the initial residual by a factor of 10^5.

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Elements/Procs.	2	4	8	16	2	4	8	16
6144	3	4	4	6	5	5	5	7
49152	3	4	4	6	4	5	5	7
393216	3	4	5	7	4	5	5	6
3145728	3	5	6	8	3	4	5	7

Table 2
Timings for the parallel solution using the stabilized FE discretization of the convection-diffusion test problem for two choices of ε: the solution times are quoted in seconds and the speed-ups are relative to the best sequential solution time.

	$\varepsilon = 10^{-2}$					$\varepsilon = 10^{-3}$				
Processors	1	2	4	8	16	1	2	4	8	16
Solution Time	770.65	484.53	347.61	228.39	136.79	688.12	442.44	277.78	187.16	108.75
Speed-Up	—	1.6	2.2	3.4	5.6	—	1.6	2.5	3.7	6.3

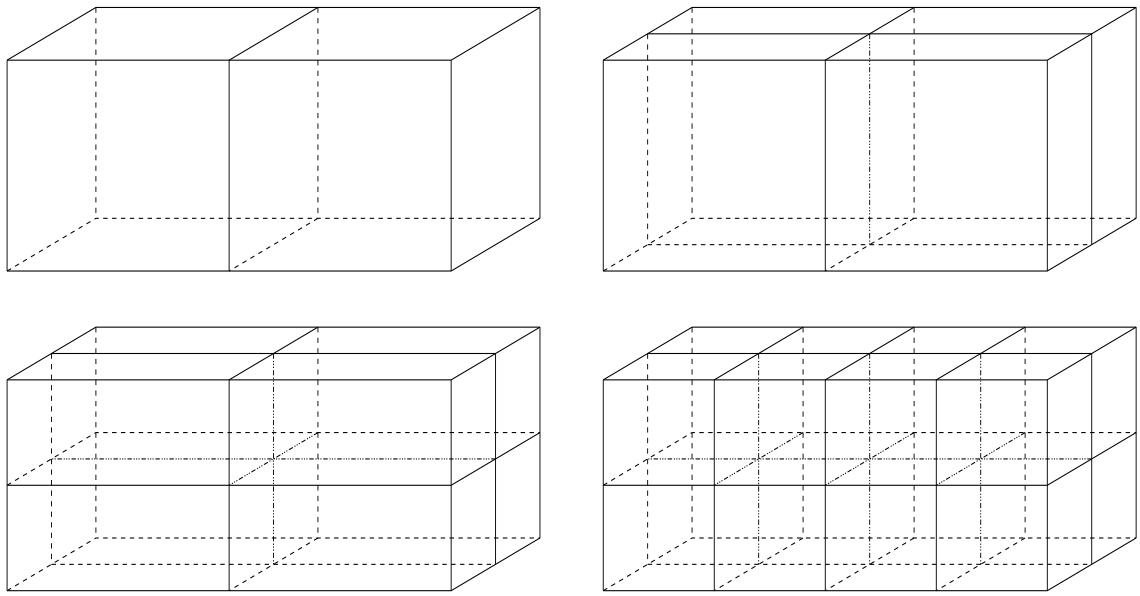
We begin with a demonstration of the quality of the weakly overlapping DD preconditioner when applied to a convection-dominated test problem of the form (1). Table 1 shows the number of preconditioned GMRES iterations that are required to solve this equation when $\underline{b}^T = (1, 0, 0)$ and f is chosen so as to permit the exact solution

$$u = \left(x - \frac{2(1 - e^{x/\varepsilon})}{(1 - e^{2/\varepsilon})} \right) y(1-y)z(1-z) \tag{12}$$

on the domain $\Omega = (0, 2) \times (0, 1) \times (0, 1)$. Two different values of ε are used, reflecting the width of the boundary layer in the solution in the region of $x = 2$. For these calculations the initial grid of 768 tetrahedral elements shown in Figure 1 (top left) is refined uniformly by up to four levels, to produce a sequence of meshes containing between 6144 and 3145782 elements.

It is clear that, as the finite element mesh is refined or the number of subdomains is increased, the number of iterations required grows extremely slowly. This is an essential property of an efficient preconditioner. In fact, the iteration counts of Table 1 suggest that the preconditioner may in fact be optimal (i.e. the condition number of the preconditioned system is bounded as the mesh is refined or the number of subdomains is increased), however we are currently unable to present any mathematical confirmation of this. In Table 2 we present timings for the complete FE calculations tabulated above on the finest mesh, with 3145728 tetrahedral elements.

Figure 2. An illustration of the partitioning strategy, based upon recursive coordinate bisection, used to obtain 2, 4, 8 and 16 subdomains in our test problem.



4. DISCUSSION

There are a number of features concerning the parallel timings in Table 2 that warrant further discussion. Perhaps the most important of these is that the algebraic action of the preconditioner that we have applied depends not only on the number of subdomains p, but also on the geometric properties of these subdomains. In each case the parallel implementation of these algorithms may be very efficient but if the algorithm itself is such that its sequential execution time is greater than that of the fastest available sequential algorithm (for this work we use [10]) then the speed-up will be adversely affected. In an effort to minimize this particular parallel overhead we have selected a simple partitioning strategy based upon recursive coordinate bisection. This strategy, illustrated in Figure 2, led to the best solution times that we were able to achieve in practice (from the small number of partitioning techniques so far considered). Furthermore, inexact solutions to the systems (11) have been used: reducing the residual by a factor of just 10 at each approximate solve (again using [10]). Whilst this can have the effect of slightly increasing the total number of iterations required for convergence it appears to yield the fastest overall solution time.

The main advantage of the partitions illustrated in Figure 2 is that the surface-area to volume ratio of the subdomains is small. This means that both the amount of additional refinement and the quantity of neighbour-to-neighbour communication is relatively small, making the cost of each iteration as low as possible. For convection-dominated problems however the number of preconditioned iterations required to converge to the solution may be decreased from those given in Table 1 by selecting a partition which contains long

thin subdomains that are aligned with the convection direction. This trade-off that exists between minimizing the number of iterations required and minimizing the cost of each iteration is an important issue that is worthy of further investigation.

In addition to the particular question of subdomain shape and the more general issue of the overall partitioning strategy there are a number of further lines of research that need to be undertaken. When a nonlinear elliptic PDE is solved for example, the FE discretization leads to a nonlinear algebraic system which may be solved using a quasi-Newton method. At each nonlinear iteration a linear Jacobian system must be dealt with, and the Jacobian matrix itself may be partitioned and assembled in parallel using the block pattern of (10). These linear systems may then be solved using the weakly overlapping preconditioner to obtain a parallel nonlinear DD algorithm. The extension to linear and nonlinear systems of PDEs may then be undertaken and assessed.

REFERENCES

1. Bank, R.E., Holst, M.: A New Paradigm for Parallel Adaptive Meshing Algorithms. SIAM J. on Sci. Comp. **22** (2000) 1411–1443.
2. Bank, R.E., Jimack, P.K.: A New Parallel Domain Decomposition Method for the Adaptive Finite Element Solution of Elliptic Partial Differential Equations. Concurrency and Computation: Practice and Experience, **13** (2001) 327–350.
3. Chan, T., Mathew, T.: Domain Decomposition Algorithms. Acta Numerica **3** (1994) 61–143.
4. Gropp, W.D., Keyes, D.E.: Parallel Performance of Domain-Decomposed Preconditioned Krylov Methods for PDEs with Locally Uniform Refinement. SIAM J. on Sci. Comp. **13** (1992) 128–145.
5. Hodgson, D.C., Jimack, P.K.: A Domain Decomposition Preconditioner for a Parallel Finite Element Solver on Distributed Unstructured Grids. Parallel Computing **23** (1997) 1157–1181.
6. Jimack, P.K., Nadeem, S.A.: A Weakly Overlapping Parallel Domain Decomposition Preconditioner for the Finite Element Solution of Elliptic Problems in Three Dimensions. In Arabnia, H.R. (ed.): Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Volume III, CSREA Press, USA (2000) 1517–1523.
7. Johnson, C.: Numerical Solutions of Partial Differential Equations by the Finite Element Method. Cambridge University Press (1987).
8. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. Int. J. Supercomputer Appl. **8** (1994) no. 3/4.
9. Oswald, P.: Multilevel Finite Element Approximation: Theory and Applications. Teubner Skripten zur Numerik, B.G. Teubner (1994).
10. Saad, Y.: SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations, Version 2. Technical Report, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, IL, USA (1994).
11. Speares, W., Berzins, M.: A 3-D Unstructured Mesh Adaptation Algorithm for Time-Dependent Shock Dominated Problems. Int. J. for Numer. Meth. in Fluids **25** (1997) 81–104.